

LiquidPi

Inferable Dependent Session Types

Dennis Griffith and Elsa L. Gunter¹

May 15, 2013

¹Supported by NASA Contract NNA10DE79C and ARO Award W911NF-09-1-0273

Introduction

- ▶ “In a concurrent system, can we statically infer temporal properties of our processes that are sensitive to the potential values communicated?”
- ▶ Our Framework
 - ▶ Currency = Pi Calculus
 - ▶ Temporal Properties = Session Types
 - ▶ Sensitivity = Limited Dependent Types

Pi Calculus (Milner)

- ▶ Process Algebra for concurrent systems
 - ▶ Channel based
 - ▶ Wraps Functional Language
- ▶ Syntax:

$$\begin{aligned} P ::= & 0 \mid P \parallel P \mid (\nu k)P \\ & \mid \text{accept } X(k).P \mid \text{request } X(k).P \\ & \mid k!(e).P \mid k?(x).P \\ & \mid k!(k).P \mid k?(k).P \\ & \mid k \triangleleft e.P_i \mid \text{case}_\tau k \Rightarrow P_i \\ & \mid \text{if } e \text{ then } P \text{ else } P \mid \text{def } X(\vec{x}; \vec{k}) = P \text{ in } P \mid X(\vec{e}, \vec{k}) \end{aligned}$$

- ▶ Semantics given by structural congruence and rewrite rules

Session Types (Honda)

- ▶ Types for Pi Calculus
- ▶ Provides guarantees:
 - ▶ Channels are used by two parties
 - ▶ Both parties agree on communication
- ▶ Syntax: $S ::= 0 \mid t \mid \mu t.S$
 $\mid !\tau.S \mid ?\tau.S$
 $\mid ![S].S \mid ?[S].S$
 $\mid \&_\tau S_i \mid \oplus_\tau S_i$
- ▶ Duality:

$$\begin{array}{llll} \overline{0} = 0 & \overline{!\tau.S} = ?\tau.\overline{S} & \overline{?\tau.S} = !\tau.\overline{S} & \overline{![S_1].S_2} = ?[S_1].\overline{S_2} \\ \overline{?[S_1].S_2} = ![S_1].\overline{S_2} & \overline{\&_\tau S_i} = \oplus_\tau \overline{S_i} & \overline{\oplus_\tau S_i} = \&_\tau \overline{S_i} \end{array}$$

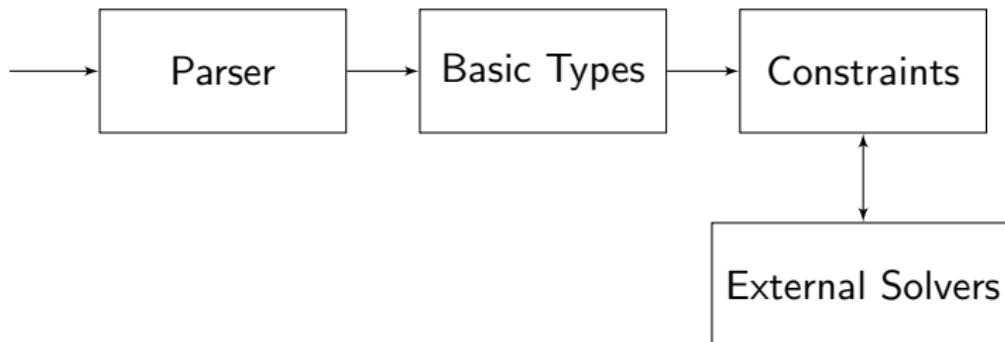
Type Systems

- ▶ Compact descriptions of behavior
- ▶ Exclude bad behavior
- ▶ Sometimes inferable
 - ▶ Efficient inference
 - ▶ Examples: ML, Haskell
- ▶ Dependent types very expressive
 - ▶ Types can depend on values
 - ▶ $\mathbb{N} \rightarrow \text{primes}$
 - ▶ Example: Coq
 - ▶ Often excludes inference

Liquid Types (Rondon)

- ▶ Want dependent types with inferability
- ▶ Restrict to leaf predicates
 - ▶ Yes: $\{\nu \in \text{int} \mid \nu \geq 5\}$, $x : \text{int} \rightarrow \{\nu \in \text{int} \mid \nu \geq x\}$
 - ▶ No: $\{\nu \in \text{int} \rightarrow \text{int} \mid \nu \text{ is bijective}\}$
- ▶ Form predicates from conjunctions of instantiated templates
- ▶ Use external solvers (SMT)
- ▶ Example: Array-bounds checking
 $\{0 \leq \nu, \star \leq \nu, \nu < \star, \nu < \text{len } \star\}$
- ▶ Induces Subtyping relation
- ▶ Path-sensitive

- ▶ Application of Liquid Typing to Session Types
- ▶ Adds expressiveness and keeps ease of use
- ▶ Extends Liquid Type subtyping to Session Types
- ▶ New type judgments
- ▶ Infer with constraint generation



Liquid Session Types

- ▶ Similar but with value binding
- ▶ Duality is similar
- ▶ Use liquid type at functional level

$$\begin{aligned}\Upsilon ::= & 0 \mid t \mid \mu t.\Upsilon \\ \mid & !\rho.\Upsilon \mid ?x \in \rho.\Upsilon \\ \mid & ![\Upsilon].\Upsilon \mid ?[\Upsilon].\Upsilon \\ \mid & \&_\tau \Upsilon_i \mid \oplus_\tau \Upsilon_i\end{aligned}$$

Selected Rules

$$\frac{\Theta; \Gamma \vdash_S P : \Delta \cdot (k : S) \quad \Gamma \vdash e : \tau}{\Theta; \Gamma \vdash_S k!(e).P : \Delta \cdot (k : !\tau.S)} \text{ T.SEND}$$

$$\frac{\Theta; \Gamma \vdash_{SL} P : \Delta \cdot k : \Upsilon \quad \Gamma \vdash_L e : \rho \quad \Gamma \vdash \rho \sqsubseteq \rho'}{\Theta; \Gamma \vdash_{SL} k!(e).P : \Delta \cdot (k : !\rho'.\Upsilon)} \text{ R.SEND}$$

where:

- ▶ \vdash_S session judgment
- ▶ \vdash functional judgment
- ▶ Θ process definitions
- ▶ Γ a functional env
- ▶ Δ channel typing env
- ▶ \vdash_{SL} is liquid session judgment
- ▶ \vdash_L is (functional) liquid judgment
- ▶ \sqsubseteq is subtyping

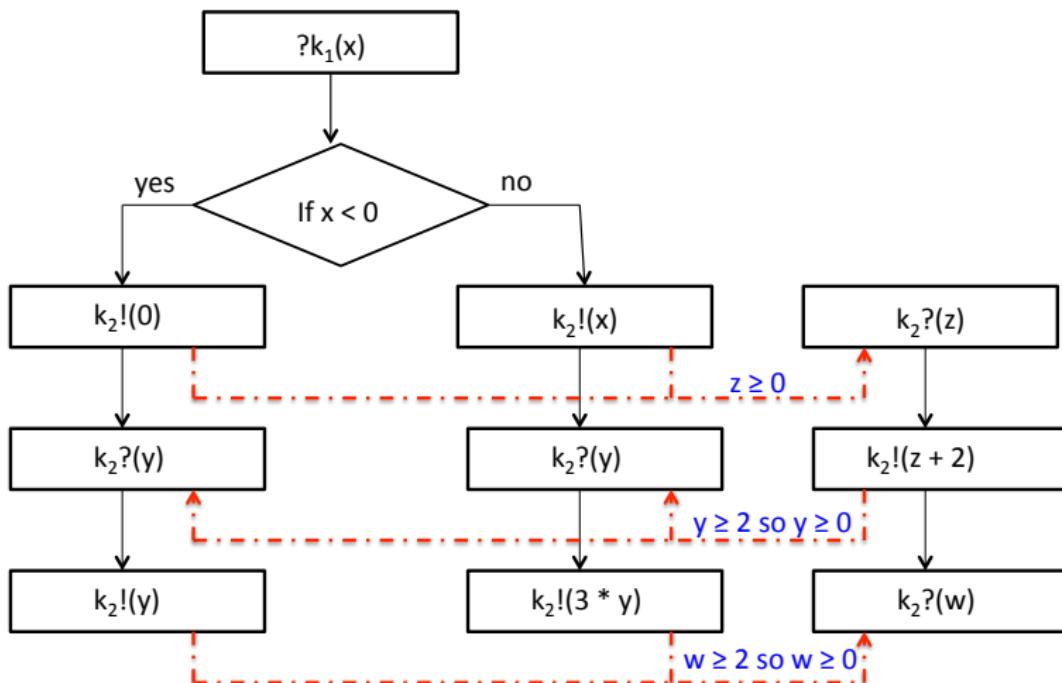
Constraints

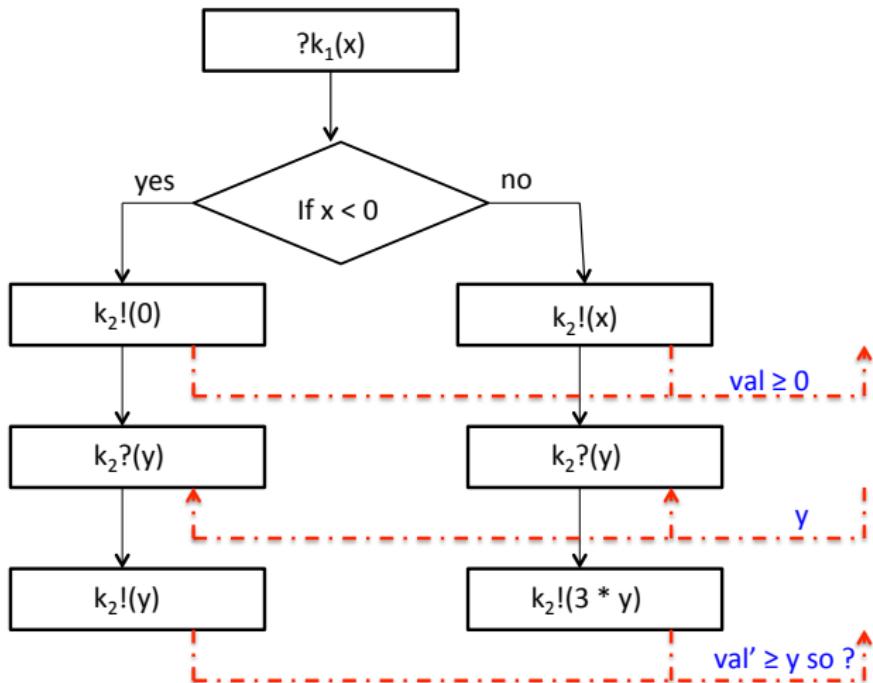
- ▶ Two constraints:
 - ▶ $\Gamma \vdash_{wf} \Upsilon$
 - ▶ $\Gamma \vdash \Upsilon_1 \sqsubseteq \Upsilon_2$
- ▶ After regular inference
- ▶ Fixed set of templates
- ▶ Constraints always have trivial solutions
- ▶ Finite solution space
- ▶ Want Maximally Informative solution
 - ▶ Naive: Try them all
 - ▶ Better: Use Iterative Weakening
 - ▶ Use SMT for efficiency

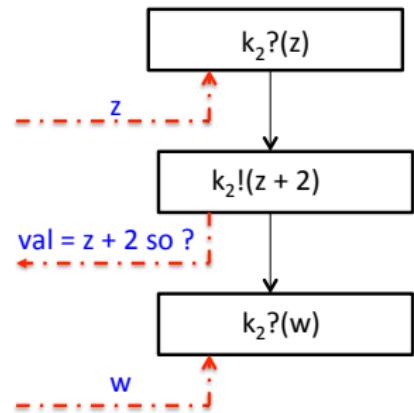
LiquidPi Example

```
?k1(x). if x < 0
    then k2!(0).k2?(y).k2!(y).0
    else k2!(x).k2?(y).k2!(3 · y).0
||k2?(z).k2!(z + 2).k2?(w).0
```

- ▶ Simple Session Types
 - ▶ $k_1 : \text{int}.0$
 - ▶ $k_2 : !\text{int}.?\text{int}.!\text{int}.0$
- ▶ Liquid Session Types: Templates = $\{\nu \in \text{int} \mid \nu \geq 0\}$
 - ▶ $k_1 : \text{int}.0$
 - ▶ $k_2 : !\{\nu \in \text{int} \mid \nu \geq 0\}.?\{\nu \in \text{int} \mid \nu \geq 0\}.!\{\nu \in \text{int} \mid \nu \geq 0\}.0$
- ▶ Selected Constraints:
 - ▶ $x < 0 \vdash \{\nu \in \text{int} \mid \nu = 0\} \sqsubseteq \{\nu \in \text{int} \mid \phi(\nu)\}$
 - ▶ $x \geq 0 \vdash \{\nu \in \text{int} \mid \nu = x\} \sqsubseteq \{\nu \in \text{int} \mid \phi(\nu)\}$







Conclusion/Future

- ▶ Need implementation (80% done)
- ▶ Need real-sized examples
- ▶ Consider Pi Calculus variants
 - ▶ Started with the simplest possible
 - ▶ Lots of interesting variants
- ▶ Should consider applying Liquid typing to any type system

Questions?